

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**  
**BỘ MÔN CÔNG NGHỆ PHẦN MỀM**



# **BÀI PROJECT\_MÔN HỌC**

## **KIẾN TRÚC VÀ THIẾT KẾ PHẦN MỀM**

**Tên sinh viên: Lê Duy Mạnh**

**Mã sinh viên: B20DCCN423**

**Lớp: D20CNPM02**

**Nhóm môn học: 02**

**Giảng viên: Trần Đình Quế**

*Hà Nội – 2024*

## LỜI CẢM ƠN

Trước tiên, em xin cảm ơn thầy Trần Đình Quế người đã tận tình hướng dẫn, cung cấp kiến thức và tạo điều kiện thuận lợi nhất cho em trong suốt quá trình thực hiện bài tập lớn này. Những ý kiến đóng góp quý báu và sự hỗ trợ nhiệt tình của thầy đã giúp chúng em thiện và phát triển ý tưởng của mình.

Em cũng xin gửi lời cảm ơn đến Ban Giám Hiệu và toàn thể giảng viên của Học viện Công nghệ Bưu chính Viễn thông và các thầy cô trong Khoa Công nghệ phần mềm đã tạo môi trường học tập và nghiên cứu thuận lợi, hỗ trợ tài liệu và trang thiết bị cần thiết cho chúng em.

Mặc dù đã cố gắng học tập và nghiên cứu trong quá trình hoàn thành bài tập lớn, tuy nhiên do kinh nghiệm còn ít, sẽ không tránh khỏi những sai sót trong bài tập lớn này, mong thầy sẽ cho những đánh giá và góp ý để những dự án tiếp theo em sẽ hoàn thành tốt hơn.

Em xin chân thành cảm ơn!

## Mục lục

LỜI CẢM ƠN .....	2
MỞ ĐẦU .....	5
Chương 1: Kiến trúc monolithic và microservice .....	9
1.1. Giới thiệu .....	9
1.2. Kiến trúc monolithic .....	9
1.3. Kiến trúc Microservice .....	10
1.3.1. Đặc điểm chính của kiến trúc microservice .....	10
1.3.2. Các công nghệ phát triển hệ microservice .....	10
1.3.3. Khi nào nên sử dụng kiến trúc Microservice .....	13
1.4. Phân rã Hệ thống phần mềm thành các service .....	14
1.5. Kết luận .....	16
Chương 2: Kiến trúc Hệ thống ecomSys dựa trên Django .....	17
2.1. Mô tả Hệ ecomSys .....	17
2.2. Kỹ thuật và Công nghệ Django .....	19
2.3. Phân rã Hệ ecomSys dựa trên Django .....	21
2.4. Kết luận .....	23
Chương 3: Thiết kế và xây dựng hệ ecomSys .....	24
3.1. Giới thiệu .....	24
3.2. Các biểu đồ đặc tả hệ ecomSys .....	25
3.2.1. Biểu đồ use case .....	25
3.2.2. Biểu đồ hoạt động/swimlane .....	29
3.2.3. Biểu đồ lớp phân tích cho từng service .....	30
3.2.4. Mô hình Data – kiểu/công nghệ dữ liệu .....	33
3.2.5. Biểu đồ lớp thiết kế cho từng service .....	34
3.3. Xây dựng hệ ecomSys .....	34
3.3.1. Module (service/project) user .....	34
3.3.2. Module Cart .....	37
3.3.3. Module search .....	38
3.3.4. Module product .....	39
3.3.5. Module payment .....	42
3.3.6. Module shipment .....	42
3.3.7. Module order .....	43
3.4. Thiết kế giao diện .....	44
3.5. Thiết kế database .....	48

<b>3.6.</b>	<b>Kết luận.....</b>	<b>48</b>
-------------	----------------------	-----------

## MỞ ĐẦU

Việc học các kiểu kiến trúc phần mềm là rất quan trọng đối với sinh viên ngành công nghệ thông tin và phát triển phần mềm. Dưới đây là một số lý do chính tại sao sinh viên cần nắm vững các kiểu kiến trúc phần mềm:

- **Hiểu Biết Về Cấu Trúc Hệ Thống:** Kiến trúc phần mềm cung cấp cái nhìn tổng quan về cấu trúc của hệ thống phần mềm, giúp sinh viên hiểu rõ cách các thành phần khác nhau tương tác và phối hợp với nhau. Điều này là cơ sở để xây dựng các hệ thống phức tạp một cách hiệu quả và có tổ chức.
- **Tối Ưu Hóa Hiệu Năng:** Các kiểu kiến trúc khác nhau có ưu điểm và nhược điểm riêng về hiệu năng. Hiểu được điều này giúp sinh viên lựa chọn và áp dụng kiến trúc phù hợp nhất để tối ưu hóa hiệu năng của hệ thống, đảm bảo hệ thống hoạt động mượt mà và đáp ứng được các yêu cầu về tốc độ và tài nguyên.
- **Tăng Cường Tính Bảo Mật:** Kiến trúc phần mềm cũng liên quan mật thiết đến vấn đề bảo mật. Một kiến trúc tốt sẽ giúp bảo vệ dữ liệu và tài nguyên của hệ thống khỏi các cuộc tấn công và xâm nhập trái phép. Sinh viên cần hiểu rõ các nguyên tắc này để thiết kế các hệ thống an toàn.
- **Dễ Dàng Bảo Trì và Nâng Cấp:** Một hệ thống có kiến trúc tốt sẽ dễ dàng bảo trì và nâng cấp. Khi các yêu cầu của người dùng thay đổi hoặc khi công nghệ phát triển, việc sửa đổi và cập nhật phần mềm sẽ trở nên dễ dàng hơn nếu hệ thống được thiết kế với kiến trúc rõ ràng và linh hoạt.
- **Tăng Tính Tái Sử Dụng:** Các thành phần phần mềm được thiết kế theo các kiểu kiến trúc chuẩn mực thường có tính tái sử dụng cao. Điều này giúp tiết kiệm thời gian và công sức khi phát triển các dự án mới, đồng thời giảm thiểu lỗi phát sinh do việc viết lại mã nguồn từ đầu.
- **Hỗ Trợ Làm Việc Nhóm:** Trong các dự án phát triển phần mềm, việc làm việc nhóm là rất quan trọng. Một kiến trúc rõ ràng giúp các thành viên trong nhóm hiểu và tuân thủ các tiêu chuẩn chung, dễ dàng phối hợp và giao tiếp với nhau, từ đó tăng hiệu quả làm việc và giảm thiểu xung đột.
- **Đáp Ứng Các Yêu Cầu Chất Lượng:** Kiến trúc phần mềm ảnh hưởng trực tiếp đến các yêu cầu chất lượng của hệ thống như khả năng mở rộng, độ tin cậy, và khả năng chịu lỗi. Sinh viên cần nắm vững kiến thức về các kiểu kiến trúc để đảm bảo rằng phần mềm được phát triển đạt các tiêu chuẩn chất lượng cần thiết.
- **Thích Ứng Với Công Nghệ Mới:** Công nghệ phần mềm luôn thay đổi và phát triển. Việc hiểu biết về các kiểu kiến trúc phần mềm giúp sinh viên dễ dàng tiếp

cận và thích ứng với các công nghệ và phương pháp mới, từ đó nâng cao năng lực chuyên môn và khả năng cạnh tranh trong nghề nghiệp.

- **Cơ Sở Cho Quyết Định Thiết Kế:** Kiến trúc phần mềm cung cấp cơ sở cho các quyết định thiết kế quan trọng trong quá trình phát triển phần mềm. Hiểu rõ các kiểu kiến trúc giúp sinh viên đưa ra các quyết định đúng đắn và hợp lý, đảm bảo rằng hệ thống được phát triển một cách hiệu quả và bền vững.
- **Nền Tảng Cho Sáng Tạo và Đổi Mới:** Cuối cùng, việc hiểu biết sâu sắc về các kiểu kiến trúc phần mềm mở ra cơ hội cho sự sáng tạo và đổi mới. Sinh viên có thể phát triển những giải pháp phần mềm độc đáo và tiên tiến, đáp ứng nhu cầu đa dạng của thị trường và góp phần thúc đẩy ngành công nghệ thông tin.

Như vậy, việc học các kiểu kiến trúc phần mềm không chỉ là một phần quan trọng của chương trình đào tạo mà còn là nền tảng vững chắc cho sự nghiệp phát triển phần mềm trong tương lai.

Kiến trúc phần mềm là một lĩnh vực rộng lớn và đóng vai trò then chốt trong việc phát triển và duy trì các hệ thống phần mềm. Có nhiều nghề nghiệp liên quan đến kiến trúc phần mềm mà sinh viên có thể theo đuổi. Dưới đây là một số nghề nghiệp tiêu biểu trong lĩnh vực này:

- **Kiến Trúc Sư Phần Mềm (Software Architect)**
  - **Mô tả công việc:** Thiết kế và xây dựng kiến trúc tổng thể cho các ứng dụng và hệ thống phần mềm, đưa ra các quyết định quan trọng về công nghệ, cấu trúc, và phương pháp phát triển.
  - **Kỹ năng cần thiết:** Kiến thức sâu rộng về các mô hình kiến trúc, kỹ năng lập trình, khả năng tư duy chiến lược, và kỹ năng giao tiếp tốt để làm việc với các nhóm phát triển và các bên liên quan.
- **Kỹ Sư Hệ Thống (Systems Engineer)**
  - **Mô tả công việc:** Thiết kế và quản lý cơ sở hạ tầng kỹ thuật cho các hệ thống phần mềm, đảm bảo tính sẵn sàng, hiệu năng và khả năng mở rộng.
  - **Kỹ năng cần thiết:** Kiến thức về phần cứng, mạng, và các dịch vụ đám mây, cùng với kỹ năng giải quyết vấn đề và khả năng làm việc nhóm.
- **Chuyên Gia DevOps (DevOps Specialist)**
  - **Mô tả công việc:** Kết nối các nhóm phát triển và vận hành để cải thiện quy trình triển khai phần mềm, sử dụng các công cụ và phương pháp tự động hóa để đảm bảo hiệu quả và liên tục.
  - **Kỹ năng cần thiết:** Kiến thức về CI/CD, quản lý cấu hình, quản lý môi trường và giám sát hệ thống.
- **Chuyên Gia Bảo Mật Phần Mềm (Software Security Specialist)**
  - **Mô tả công việc:** Đảm bảo rằng hệ thống phần mềm được thiết kế và triển khai với các biện pháp bảo mật hiệu quả, giảm thiểu các lỗ hổng bảo mật.

- **Kỹ năng cần thiết:** Kiến thức về bảo mật phần mềm, các phương pháp tấn công và phòng thủ, và khả năng đánh giá và cải thiện các biện pháp bảo mật.
- Kỹ Sư Kiểm Thử Phần Mềm (Software Test Engineer)
  - **Mô tả công việc:** Thiết kế và thực hiện các kế hoạch kiểm thử, viết và chạy các test case, báo cáo lỗi và phối hợp với nhóm phát triển để khắc phục lỗi.
  - **Kỹ năng cần thiết:** Kiến thức về các phương pháp kiểm thử (manual và automated), kỹ năng lập trình cơ bản, và khả năng phân tích.

Kiến trúc hướng dịch vụ (SOA) là một phương pháp thiết kế phần mềm trong đó các ứng dụng được xây dựng như một loạt các dịch vụ độc lập, có thể tái sử dụng và có khả năng tích hợp tốt với nhau qua các dịch vụ tiêu chuẩn. Trong môi trường Django, một framework nổi tiếng và phổ biến cho phát triển web, các dự án sử dụng kiến trúc hướng dịch vụ để tạo ra các ứng dụng linh hoạt, dễ mở rộng và dễ bảo trì. Ví dụ một dự án e-commerce có thể sử dụng kiến trúc hướng dịch vụ để quản lý các chức năng như quản lý sản phẩm, quản lý đơn hàng và thanh toán.

Việc tích hợp AI và deep learning vào các hệ thống như e-commerce, hệ quản lý nhân sự, thư viện, giao thông, và quản lý dịch vụ mang lại nhiều lợi ích và cải thiện hiệu suất hoạt động. Dưới đây là một số yêu cầu và lý do cụ thể cho từng lĩnh vực:

#### 1. E-commerce (Thương mại điện tử)

- **Cá nhân hóa trải nghiệm người dùng:** Sử dụng AI để phân tích hành vi mua sắm và đề xuất sản phẩm phù hợp với từng khách hàng. Deep learning có thể tạo ra các mô hình dự đoán dựa trên lịch sử mua sắm và sở thích cá nhân.
- **Tối ưu hóa tìm kiếm:** Cải thiện khả năng tìm kiếm sản phẩm bằng cách sử dụng AI để hiểu ngữ cảnh và ý định của người dùng, cung cấp kết quả tìm kiếm chính xác hơn.
- Dự đoán xu hướng mua sắm và nhu cầu thị trường, giúp quản lý kho hàng và chiến lược kinh doanh hiệu quả hơn.
- Sử dụng chatbot AI để cung cấp dịch vụ hỗ trợ khách hàng 24/7, giải quyết các câu hỏi thường gặp và hỗ trợ quá trình mua hàng.

#### 2. Hệ quản lý nhân sự (Human Resource Management System - HRMS)

- **Tuyển dụng thông minh:** Sử dụng AI để sàng lọc hồ sơ ứng viên, đánh giá kỹ năng và phù hợp với vị trí tuyển dụng, giảm bớt công việc thủ công và tăng tính hiệu quả.
- **Quản lý hiệu suất nhân viên:** Phân tích dữ liệu về hiệu suất làm việc của nhân viên để đưa ra các đánh giá chính xác và cá nhân hóa kế hoạch phát triển nghề nghiệp.

- **Dự đoán rời bỏ nhân viên:** Sử dụng mô hình AI để dự đoán khả năng nhân viên sẽ rời bỏ công ty, từ đó có các biện pháp giữ chân kịp thời.
- **Tự động hóa quy trình hành chính:** Tự động hóa các tác vụ hành chính như chấm công, tính lương, và quản lý phúc lợi, giúp giảm bớt gánh nặng công việc và tăng hiệu suất.

### 3. Thư viện

- **Quản lý tài liệu thông minh:** Sử dụng AI để phân loại và sắp xếp tài liệu, giúp việc tìm kiếm và truy cập tài liệu dễ dàng hơn.
- **Đề xuất tài liệu:** Dựa trên lịch sử mượn sách và sở thích của người dùng, AI có thể đề xuất các tài liệu, sách, hoặc bài báo phù hợp
- **Phân tích dữ liệu người dùng:** Thu thập và phân tích dữ liệu về hành vi của người dùng để cải thiện dịch vụ và đáp ứng nhu cầu của độc giả
- **Bảo quản và số hóa tài liệu:** Sử dụng công nghệ AI để nhận diện và số hóa các tài liệu cũ, giúp bảo quản và truy cập tài liệu dễ dàng hơn

### 4. Giao thông

- **Quản lý giao thông thông minh:** Sử dụng AI để phân tích dữ liệu từ các cảm biến giao thông, camera, và các nguồn khác để điều tiết giao thông, giảm thiểu tắc nghẽn và tai nạn.
- **Dự đoán và lên kế hoạch tuyến đường:** AI có thể dự đoán lưu lượng giao thông và đề xuất các tuyến đường tối ưu cho người dùng.
- **Quản lý phương tiện công cộng:** Tối ưu hóa lịch trình và lộ trình của các phương tiện giao thông công cộng dựa trên dữ liệu về hành khách và lưu lượng giao thông
- **Hệ thống giao thông tự hành:** Phát triển và triển khai các phương tiện tự hành sử dụng deep learning để nhận diện môi trường xung quanh và điều khiển phương tiện

Báo cáo này nhằm mục đích trình bày về các kiến trúc trong thiết kế phần mềm và Thiết kế xây dựng hệ thống ecomSys Nội dung bao gồm:

- Kiến trúc monolithic và microservice
- Kiến trúc Hệ thống ecomSys dựa trên Django
- Thiết kế và xây dựng hệ ecomSys

Ngoài Mở đầu và Kết luận Báo cáo được cấu trúc thành 3 Chương như sau:

**Chương 1: Kiến trúc monolithic và microservice**

**Chương 2: Kiến trúc Hệ thống ecomSys dựa trên Django**

**Chương 3: Thiết kế và xây dựng hệ ecomSys**



## Chương 1: Kiến trúc monolithic và microservice

### 1.1. Giới thiệu

Kiến trúc phần mềm đóng vai trò quan trọng trong việc thiết kế và phát triển các ứng dụng. Hai mô hình kiến trúc phổ biến nhất hiện nay là kiến trúc Monolithic và kiến trúc Microservice. Mỗi mô hình có những ưu điểm và nhược điểm riêng, phù hợp với các loại dự án và nhu cầu khác nhau. Dưới đây là cái nhìn tổng quan về hai kiểu kiến trúc này:

#### 1.2. Kiến trúc monolithic

Giới thiệu chung: Kiến trúc Monolithic là một mô hình truyền thống trong đó toàn bộ ứng dụng được xây dựng như một khối duy nhất. Tất cả các thành phần của ứng dụng như giao diện người dùng, logic nghiệp vụ, và cơ sở dữ liệu đều được kết hợp thành một ứng dụng đơn lẻ.

Đặc điểm:

- **Đơn nhất:** Ứng dụng là một khối duy nhất, tất cả các thành phần được triển khai cùng nhau.
- **Triển khai dễ dàng:** Vì chỉ có một khối duy nhất, việc triển khai ứng dụng đơn giản hơn.
- **Đơn giản hóa phát triển:** Mô hình này thường dễ hiểu và dễ phát triển cho các dự án nhỏ hoặc đội ngũ phát triển nhỏ.
- **Hiệu suất:** Do tất cả các thành phần chạy trong cùng một không gian bộ nhớ, giao tiếp giữa các thành phần có thể rất nhanh.

Nhược điểm:

- **Khó mở rộng:** Khi ứng dụng phát triển lớn, việc quản lý và mở rộng trở nên phức tạp.
- **Khả năng bảo trì kém:** Mỗi lần thay đổi mã, toàn bộ ứng dụng phải được xây dựng và triển khai lại, dẫn đến rủi ro cao về lỗi.

- **Giới hạn công nghệ:** Khó sử dụng nhiều công nghệ hoặc ngôn ngữ lập trình khác nhau cho các thành phần khác nhau của ứng dụng.
- **Khó khăn trong việc triển khai và phát triển theo nhóm:** Khi nhiều nhà phát triển làm việc trên cùng một mã nguồn, việc quản lý mã và tránh xung đột trở nên khó khăn.

### 1.3. Kiến trúc Microservice

Kiến trúc Microservice là một mô hình hiện đại, trong đó ứng dụng được chia thành các dịch vụ nhỏ, độc lập, mỗi dịch vụ thực hiện một chức năng cụ thể và có thể được phát triển, triển khai và mở rộng riêng lẻ.

#### 1.3.1. Đặc điểm chính của kiến trúc microservice

- **Tính Độc Lập:**
  - Mỗi dịch vụ là một đơn vị độc lập, có thể phát triển, triển khai và mở rộng riêng rẽ mà không ảnh hưởng đến các dịch vụ khác.
  - Các dịch vụ giao tiếp với nhau thông qua các giao thức nhẹ nhàng như HTTP/REST hoặc gRPC.
- **Tách Biệt về Chức Năng:**
  - Mỗi microservice chịu trách nhiệm cho một chức năng cụ thể của ứng dụng, chẳng hạn như quản lý người dùng, xử lý đơn hàng, hoặc thanh toán.
  - Điều này giúp giảm thiểu sự phụ thuộc giữa các phần của ứng dụng, làm cho hệ thống tổng thể dễ quản lý hơn.
- **Khả Năng Mở Rộng:**
  - Các dịch vụ có thể được mở rộng độc lập dựa trên nhu cầu thực tế. Ví dụ, nếu dịch vụ xử lý thanh toán gặp nhiều lưu lượng, chỉ cần mở rộng dịch vụ này mà không ảnh hưởng đến các dịch vụ khác.
- **Tự Do Công Nghệ:**
  - Mỗi dịch vụ có thể được xây dựng bằng các ngôn ngữ lập trình và công nghệ khác nhau, tùy thuộc vào yêu cầu cụ thể và sự tối ưu hóa cho từng dịch vụ.
- **Độc Lập về Triển Khai:**
  - Các dịch vụ có thể được triển khai độc lập. Điều này cho phép các đội ngũ phát triển phát hành các tính năng mới hoặc sửa lỗi mà không cần phải triển khai lại toàn bộ ứng dụng.

#### 1.3.2. Các công nghệ phát triển hệ microservice

Để phát triển và triển khai hệ thống microservice hiệu quả, nhiều công nghệ và công cụ khác nhau có thể được sử dụng. Dưới đây là các công nghệ phổ biến và các công cụ quan trọng trong việc xây dựng, triển khai và quản lý hệ thống microservice:

- Ngôn ngữ lập trình

Các ngôn ngữ lập trình phổ biến cho việc phát triển microservice bao gồm:

- **Java:** Spring Boot, Micronaut
- **JavaScript/TypeScript:** Node.js, NestJS
- **Python:** Flask, Django, FastAPI
- **Go:** Go Kit, Go Micro
- **C#:** .NET Core
- **Ruby:** Ruby on Rails
- **Kotlin:** Ktor

- Frameworks và Libraries

Các framework và libraries giúp đơn giản hóa việc phát triển microservice:

- **Spring Boot (Java):** Cung cấp một nền tảng mạnh mẽ và giàu tính năng để phát triển microservice.
- **Micronaut (Java, Kotlin, Groovy):** Thiết kế để xây dựng các ứng dụng microservice nhẹ và nhanh.
- **Node.js (JavaScript/TypeScript):** Phù hợp cho các ứng dụng I/O cao.
- **NestJS (TypeScript):** Xây dựng các ứng dụng microservice mô-đun và có cấu trúc rõ ràng.
- **Flask/FastAPI (Python):** Phát triển các API nhanh chóng và dễ dàng.
- **Go Kit, Go Micro (Go):** Các công cụ mạnh mẽ để xây dựng microservice bằng Go

- Containerization

Containerization giúp đóng gói ứng dụng và các dependencies của nó vào một container đơn lẻ, đảm bảo tính nhất quán và dễ triển khai:

- **Docker:** Nền tảng container phổ biến nhất, giúp xây dựng, triển khai và chạy các ứng dụng trong container.
- **Podman:** Một công cụ container tương tự như Docker, với một số ưu điểm về bảo mật và tính năng.

- Orchestration

Orchestration giúp quản lý và điều phối nhiều container:

- **Kubernetes:** Nền tảng phổ biến nhất để quản lý container, cung cấp khả năng mở rộng tự động, phục hồi tự động và triển khai dễ dàng.

- **Docker Swarm:** Công cụ orchestration tích hợp của Docker, dễ sử dụng và phù hợp cho các hệ thống nhỏ hơn.
- **Apache Mesos:** Một giải pháp mạnh mẽ cho việc quản lý tài nguyên trong các môi trường lớn.
- **API Gateway**  
API Gateway quản lý các request từ client và cung cấp các dịch vụ như routing, load balancing, và security:
  - **Kong:** API Gateway mã nguồn mở và có thể mở rộng.
  - **NGINX:** Được sử dụng rộng rãi làm API Gateway và reverse proxy.
  - **Zuul (Netflix OSS):** Một API Gateway do Netflix phát triển, tích hợp tốt với hệ sinh thái Spring.
- **Service Discovery**  
Service Discovery cho phép các dịch vụ tìm và giao tiếp với nhau:
  - **Consul:** Cung cấp service discovery, health checking, và cấu hình phân tán.
  - **Eureka (Netflix OSS):** Một service registry để microservice tự động đăng ký và phát hiện lẫn nhau.
  - **Etdcd:** Một kho dữ liệu key-value phân tán, được sử dụng cho service discovery.
- **Messaging và Event Streaming**  
Để các dịch vụ giao tiếp với nhau thông qua message hoặc event:
  - **RabbitMQ:** Message broker mã nguồn mở, hỗ trợ nhiều giao thức messaging.
  - **Apache Kafka:** Một nền tảng streaming phân tán, rất phổ biến cho việc xử lý dữ liệu theo thời gian thực.
  - **NATS:** Hệ thống messaging đơn giản, hiệu suất cao.
- **Monitoring và Logging**  
Giám sát và ghi log giúp theo dõi hoạt động và hiệu suất của các dịch vụ:
  - **Prometheus:** Hệ thống giám sát và cảnh báo mã nguồn mở.
  - **Grafana:** Công cụ phân tích và hiển thị dữ liệu, thường được sử dụng cùng với Prometheus.
  - **ELK Stack (Elasticsearch, Logstash, Kibana):** Bộ công cụ ghi log và phân tích log mạnh mẽ.
  - **Jaeger:** Công cụ tracing mã nguồn mở để theo dõi các yêu cầu qua nhiều dịch vụ.
- **CI/CD**  
Continuous Integration và Continuous Deployment giúp tự động hóa quy trình phát triển và triển khai:
  - **Jenkins:** Công cụ CI/CD mã nguồn mở phổ biến.

- **GitLab CI/CD**: Tích hợp sẵn trong GitLab, hỗ trợ toàn bộ quy trình DevOps.
- **CircleCI**: Dịch vụ CI/CD mạnh mẽ và dễ sử dụng
- **Travis CI**: Dịch vụ CI/CD mã nguồn mở, tích hợp tốt với GitHub
- Security

Bảo mật cho các microservice là yếu tố rất quan trọng:

- **OAuth2**: Giao thức chuẩn cho authorization.
- **OpenID Connect**: Lớp nhận dạng xây dựng trên OAuth2
- **JWT (JSON Web Tokens)**: Định dạng token thường được sử dụng để truyền tải thông tin giữa các dịch vụ một cách an toàn.
- **Istio**: Một service mesh giúp quản lý giao tiếp, bảo mật và giám sát giữa các microservice
- **Kết luận**  
Việc lựa chọn các công nghệ và công cụ phát triển microservice phụ thuộc vào nhu cầu cụ thể của dự án, kỹ năng của đội ngũ phát triển, và yêu cầu hạ tầng. Sự kết hợp đúng đắn giữa các công cụ này sẽ giúp xây dựng và vận hành một hệ thống microservice hiệu quả và linh hoạt.

### 1.3.3. Khi nào nên sử dụng kiến trúc Microservice

Kiến trúc microservice không phải lúc nào cũng là lựa chọn tối ưu cho mọi dự án. Việc quyết định sử dụng kiến trúc microservice phụ thuộc vào nhiều yếu tố như quy mô, yêu cầu kỹ thuật, và khả năng quản lý của tổ chức. Dưới đây là những trường hợp nên xem xét áp dụng kiến trúc microservice:

- Ứng dụng lớn và phức tạp
  - **Quy mô và độ phức tạp cao**: Khi ứng dụng có quy mô lớn với nhiều module chức năng phức tạp, microservice giúp phân tách ứng dụng thành các phần nhỏ, dễ quản lý hơn.
  - **Phát triển và bảo trì dễ dàng hơn**: Các nhóm phát triển có thể làm việc độc lập trên từng dịch vụ, giảm thiểu sự phụ thuộc và tăng tốc độ phát triển và triển khai.
- Nhu cầu mở rộng linh hoạt
  - **Khả năng mở rộng theo nhu cầu**: Với microservice, bạn có thể mở rộng từng phần của ứng dụng một cách độc lập dựa trên nhu cầu thực tế, thay vì phải mở rộng toàn bộ ứng dụng như kiến trúc monolithic.
  - **Tối ưu hóa tài nguyên**: Cho phép phân bổ tài nguyên hợp lý hơn, sử dụng hiệu quả tài nguyên máy chủ và giảm chi phí.
- Đòi hỏi tính linh hoạt trong công nghệ

- **Sử dụng công nghệ khác nhau:** Microservice cho phép các nhóm phát triển sử dụng các công nghệ, ngôn ngữ lập trình, và frameworks khác nhau phù hợp nhất cho từng dịch vụ cụ thể.
- **Công nghệ cập nhật và thử nghiệm:** Dễ dàng thử nghiệm và áp dụng các công nghệ mới mà không ảnh hưởng đến toàn bộ hệ thống.
- Yêu cầu về thời gian triển khai và cập nhật nhanh chóng
  - **Triển khai và cập nhật độc lập:** Các dịch vụ có thể được triển khai và cập nhật độc lập mà không gây gián đoạn toàn bộ hệ thống, giúp giảm thời gian gián đoạn dịch vụ và tăng tốc độ phát triển.
  - **Phát hành liên tục (CI/CD):** Dễ dàng triển khai các quy trình CI/CD, tự động hóa quy trình phát triển và triển khai, giảm thời gian đưa sản phẩm ra thị trường.
- Tăng cường khả năng chịu lỗi và độ tin cậy
  - **Cách ly lỗi:** Nếu một dịch vụ gặp sự cố, chỉ dịch vụ đó bị ảnh hưởng, các phần còn lại của hệ thống vẫn hoạt động bình thường.
  - **Khả năng phục hồi nhanh chóng:** Các dịch vụ có thể được tái khởi động và khôi phục độc lập, giảm thiểu thời gian gián đoạn.
- Phát triển và triển khai bởi nhiều đội ngũ
  - **Quản lý đội ngũ phân tán:** Khi có nhiều đội ngũ phát triển làm việc trên các phần khác nhau của ứng dụng, microservice giúp tách biệt trách nhiệm và giảm thiểu xung đột giữa các đội ngũ.
  - **Quản lý dự án hiệu quả:** Các nhóm có thể làm việc trên các dịch vụ riêng lẻ, tăng cường sự tập trung và chuyên môn hóa.
- Yêu cầu bảo mật và tuân thủ
  - **Tăng cường bảo mật:** Mỗi dịch vụ có thể có các biện pháp bảo mật riêng, tăng cường bảo mật cho toàn bộ hệ thống.
  - **Tuân thủ quy định:** Dễ dàng hơn trong việc đảm bảo tuân thủ các quy định và tiêu chuẩn khác nhau cho từng phần của ứng dụng.

#### 1.4. Phân rã Hệ thống phần mềm thành các service

Phân rã hệ thống phần mềm thành các service là một bước quan trọng trong việc thiết kế kiến trúc microservice. Việc này đòi hỏi hiểu rõ các thành phần và chức năng của hệ thống để có thể phân chia chúng thành các dịch vụ độc lập, có thể triển khai và mở rộng riêng rẽ. Dưới đây là các bước và phương pháp để phân rã hệ thống phần mềm thành các service:

- Xác định các miền chức năng (Domain)
  - Bước 1: Hiểu rõ về hệ thống và các yêu cầu kinh doanh
    - **Phân tích nghiệp vụ:** Tìm hiểu kỹ về các yêu cầu kinh doanh, quy trình nghiệp vụ và các chức năng chính của hệ thống.
    - **Tương tác với các bên liên quan:** Làm việc với các bên liên quan như khách hàng, người dùng cuối, và các chuyên gia nghiệp vụ để hiểu rõ các yêu cầu và mục tiêu của hệ thống.

- Bước 2: Phân tích các miền chức năng
  - **Domain-Driven Design (DDD):** Sử dụng phương pháp DDD để xác định các miền chức năng chính của hệ thống. DDD giúp chia hệ thống thành các vùng ranh giới rõ ràng (Bounded Context), mỗi vùng có một mô hình nghiệp vụ riêng.
  - **Chia nhỏ chức năng:** Xác định và phân chia các chức năng chính của hệ thống theo từng miền nghiệp vụ. Ví dụ, trong một hệ thống e-commerce, các miền chức năng có thể bao gồm quản lý sản phẩm, quản lý đơn hàng, thanh toán, và quản lý người dùng
- Xác định các service
  - Bước 3: Định nghĩa các service
    - **Phân rã các miền chức năng:** Chia nhỏ các miền chức năng thành các service nhỏ hơn. Mỗi service nên đảm nhiệm một chức năng cụ thể và độc lập.
    - **Nguyên tắc Single Responsibility:** Mỗi service nên có một trách nhiệm duy nhất và rõ ràng. Điều này giúp tăng cường khả năng bảo trì và mở rộng.
  - Bước 4: Định nghĩa giao tiếp giữa các service
    - **API và giao thức giao tiếp:** Xác định cách các service sẽ giao tiếp với nhau. Thông thường, các service giao tiếp qua các API RESTful hoặc gRPC.
    - **Quản lý phụ thuộc:** Giảm thiểu sự phụ thuộc giữa các service để tránh tình trạng một service bị lỗi gây ảnh hưởng đến các service khác.
- Thiết kế và triển khai các service
  - Bước 5: Thiết kế chi tiết cho từng service
    - **Lựa chọn công nghệ:** Chọn công nghệ và ngôn ngữ lập trình phù hợp cho từng service. Mỗi service có thể sử dụng công nghệ khác nhau dựa trên yêu cầu cụ thể của nó.
    - **Thiết kế cơ sở dữ liệu:** Quyết định xem mỗi service có nên có cơ sở dữ liệu riêng hay không. Thường thì mỗi service nên có một cơ sở dữ liệu riêng biệt để đảm bảo tính độc lập và khả năng mở rộng.
  - Bước 6: Xây dựng và triển khai các service
    - **Containerization:** Sử dụng Docker để đóng gói các service trong container, đảm bảo tính nhất quán và dễ triển khai.
    - **Orchestration:** Sử dụng Kubernetes hoặc các công cụ orchestration khác để quản lý việc triển khai, mở rộng và giám sát các service.
- Quản lý và tối ưu hóa hệ thống
  - Bước 7: Giám sát và logging

- **Công cụ giám sát:** Sử dụng các công cụ như Prometheus và Grafana để giám sát hiệu suất và sức khỏe của các service.
- **Logging và tracing:** Sử dụng ELK Stack hoặc Jaeger để thu thập và phân tích log, giúp theo dõi các request qua các service và xác định lỗi.
- Bước 8: Bảo mật và quản lý lỗi
  - **API Gateway:** Sử dụng API Gateway để quản lý truy cập, cung cấp bảo mật và kiểm soát các request từ client
  - **Circuit Breaker:** Sử dụng các pattern như Circuit Breaker để tăng khả năng chịu lỗi và đảm bảo hệ thống vẫn hoạt động khi một service bị lỗi.

## 1.5. Kết luận

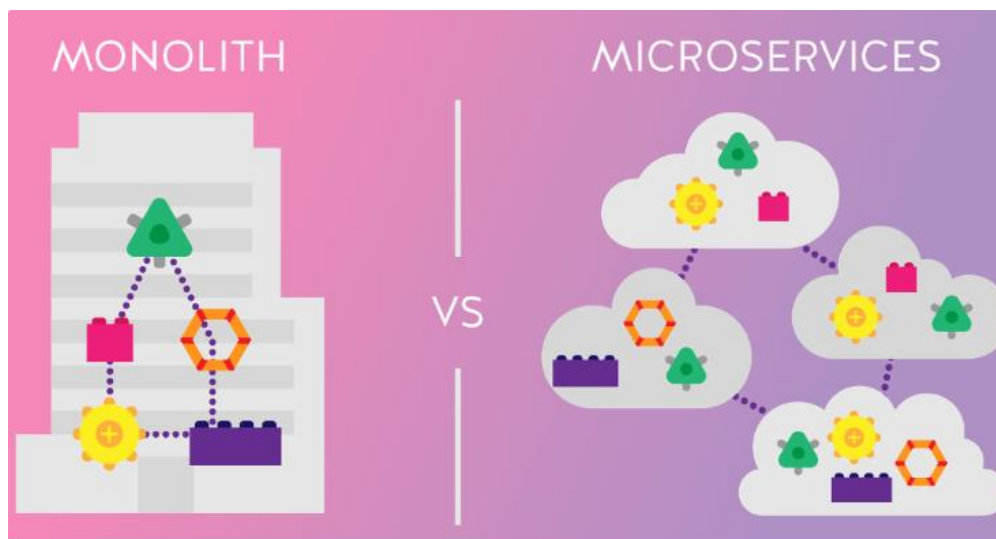
So sánh Monolithic và Microservice

Tiêu chí	Kiến trúc Monolithic	Kiến trúc Microservice
Phát triển và triển khai	Đơn giản, một khối duy nhất	Phức tạp, nhiều dịch vụ độc lập
Mở rộng	Khó mở rộng khi ứng dụng lớn	Dễ mở rộng từng dịch vụ cụ thể
Quản lý	Dễ quản lý cho các ứng dụng nhỏ	Quản lý phức tạp, cần hệ thống quản lý dịch vụ
Bảo trì	Khó bảo trì khi ứng dụng lớn	Dễ bảo trì, từng dịch vụ có thể bảo trì riêng
Công nghệ	Giới hạn trong một công nghệ hoặc ngôn ngữ	Tự do sử dụng nhiều công nghệ, ngôn ngữ khác nhau

- **Kiến trúc Monolithic** thích hợp cho các ứng dụng đơn giản, có quy mô nhỏ hoặc yêu cầu hiệu suất cao trong môi trường mạng nội bộ.



- **Kiến trúc Microservice** thích hợp cho các ứng dụng lớn, phức tạp, yêu cầu tính linh hoạt và khả năng mở rộng cao, đặc biệt là trong môi trường phân tán và môi trường cloud.



## Chương 2: Kiến trúc Hệ thống ecomSys dựa trên Django

### 2.1. Mô tả Hệ ecomSys

Hệ thống e-commerce (ecomSys) là một nền tảng trực tuyến cho việc mua bán hàng hóa và dịch vụ. Đây là một môi trường kinh doanh trực tuyến phức tạp, kết nối người mua và người bán thông qua giao diện web và ứng dụng di động. Dưới đây là mô tả của ecomSys bằng ngôn ngữ tự nhiên:

Hệ thống e-commerce bao gồm nhiều thành phần chức năng cốt lõi để hỗ trợ quá trình mua bán trực tuyến. Các thành phần này được tổ chức thành các dịch vụ độc lập để tạo ra một hệ thống phân tán linh hoạt và dễ mở rộng.

#### 1. Dịch vụ Quản lý Sản phẩm (Product Management Service):

- Dịch vụ này chịu trách nhiệm quản lý thông tin về sản phẩm trên nền tảng e-commerce.
- Nó cung cấp chức năng cho việc thêm, sửa đổi và xóa sản phẩm, cũng như quản lý danh mục sản phẩm và các thuộc tính của chúng.

#### 2. Dịch vụ Quản lý Đơn hàng (Order Management Service):

- Dịch vụ này điều phối các hoạt động liên quan đến quản lý đơn hàng của khách hàng.
- Nó xử lý các yêu cầu đặt hàng, xác nhận đơn hàng, cập nhật trạng thái và quản lý thông tin vận chuyển.

### **3. Dịch vụ Thanh toán (Payment Service):**

- Dịch vụ này xử lý các giao dịch thanh toán an toàn và đáng tin cậy.
- Nó tích hợp với các cổng thanh toán và dịch vụ tài chính để xử lý thanh toán từ khách hàng.

### **4. Dịch vụ Quản lý Người dùng (User Management Service):**

- Dịch vụ này quản lý thông tin về người dùng và tài khoản trên nền tảng e-commerce.
- Nó xử lý việc đăng ký, đăng nhập, quản lý thông tin cá nhân và xác thực người dùng.

### **5. Dịch vụ Quản lý Giỏ hàng (Cart Management Service):**

- Dịch vụ này quản lý các mục hàng được thêm vào giỏ hàng của người dùng.
- Nó cho phép người dùng thêm, cập nhật hoặc xóa các mục hàng trong giỏ hàng của họ.

### **6. Dịch vụ Thống kê và Phân tích (Analytics Service):**

- Dịch vụ này cung cấp các báo cáo thống kê và phân tích về hoạt động mua bán trên nền tảng e-commerce.
- Nó thu thập và phân tích dữ liệu về các giao dịch, hành vi mua sắm của người dùng và xu hướng thị trường để hỗ trợ quyết định kinh doanh.

### **7. Dịch vụ Quản lý Đánh giá và Phản hồi (Review Management Service):**

- Dịch vụ này quản lý các đánh giá và phản hồi từ phía khách hàng về sản phẩm và dịch vụ trên nền tảng e-commerce.
- Nó cho phép người dùng đăng và xem các đánh giá, cũng như gửi phản hồi về sản phẩm và trải nghiệm mua sắm của họ.

Hệ thống e-commerce này được xây dựng với một kiến trúc microservice linh hoạt và dễ mở rộng. Mỗi dịch vụ độc lập thực hiện một nhiệm vụ cụ thể, giúp tăng cường tính linh hoạt và hiệu suất của hệ thống.

## 2.2. Kỹ thuật và Công nghệ Django

Kỹ thuật và Công nghệ Django là một phần quan trọng trong việc phát triển ứng dụng web với Python. Django là một framework phát triển web mạnh mẽ, linh hoạt và dễ sử dụng, được xây dựng trên ngôn ngữ lập trình Python. Dưới đây là một mô tả chi tiết về kỹ thuật và công nghệ liên quan đến Django:

### 1. Django Framework:

#### 1.1. Mô hình MTV:

- Django sử dụng mô hình MTV (Model-Template-View) cho việc phát triển ứng dụng web.
- **Model:** Định nghĩa cấu trúc dữ liệu và quan hệ với cơ sở dữ liệu.
- **Template:** Xây dựng giao diện người dùng.
- **View:** Xử lý logic và tương tác với dữ liệu.

#### 1.2. ORM (Object-Relational Mapping):

- Django cung cấp một ORM mạnh mẽ, cho phép tương tác với cơ sở dữ liệu bằng cách sử dụng các đối tượng Python thay vì SQL truyền thống.
- Giúp giảm thiểu việc viết mã SQL thủ công và làm cho việc thao tác với cơ sở dữ liệu trở nên dễ dàng hơn.

#### 1.3. Admin Interface:

- Django cung cấp một giao diện quản trị tích hợp sẵn, cho phép quản lý dữ liệu từ cơ sở dữ liệu một cách dễ dàng và nhanh chóng.
- Người dùng có thể tạo, đọc, cập nhật và xóa dữ liệu mà không cần viết mã nguồn.

## 2. Công nghệ và Kỹ thuật liên quan:

### 2.1. Python:

- Django được xây dựng trên ngôn ngữ lập trình Python, một ngôn ngữ dễ học, dễ đọc và mạnh mẽ.
- Python giúp Django trở thành một framework phát triển web phổ biến và được ưa chuộng.

### 2.2. HTML, CSS, JavaScript:

- Django sử dụng HTML, CSS và JavaScript để tạo ra giao diện người dùng đẹp và tương tác.
- Django hỗ trợ việc tích hợp các framework front-end như Bootstrap hoặc Vue.js để phát triển giao diện người dùng phong phú và đáp ứng.

### **2.3. PostgreSQL, MySQL, SQLite:**

- Django hỗ trợ nhiều loại cơ sở dữ liệu phổ biến như PostgreSQL, MySQL và SQLite.
- Người dùng có thể lựa chọn cơ sở dữ liệu phù hợp với yêu cầu của ứng dụng và triển khai.

### **2.4. RESTful API:**

- Django REST Framework là một công cụ mạnh mẽ cho việc xây dựng API RESTful trong Django.
- Cho phép phát triển các ứng dụng web có khả năng mở rộng và tương tác với các dịch vụ khác.

### **2.5. Docker:**

- Docker cung cấp một cách tiện lợi để triển khai ứng dụng Django trong một môi trường container hóa.
- Giúp giảm thiểu sự phụ thuộc vào môi trường và tăng tính di động của ứng dụng.

### **2.6. Testing:**

- Django cung cấp một hệ thống testing tích hợp sẵn cho việc kiểm tra và đảm bảo chất lượng của mã nguồn.
- Bao gồm các loại tests như unit tests, functional tests và integration tests.

### **2.7. Deployment:**

- Django có nhiều cách triển khai ứng dụng, bao gồm triển khai trên máy chủ riêng, dịch vụ đám mây như Heroku hoặc AWS, và sử dụng các công cụ quản lý hệ thống như Docker Swarm hoặc Kubernetes.

### **Kết luận:**

Django là một framework phát triển web mạnh mẽ và linh hoạt, được xây dựng trên ngôn ngữ Python. Với sự hỗ trợ của các công nghệ và kỹ thuật tiên tiến, Django giúp phát triển ứng dụng web nhanh chóng, hiệu quả và đáng tin cậy.

## 2.3. Phân rã Hệ ecomSys dựa trên Django

Phân rã hệ thống ecomSys dựa trên Django sẽ tập trung vào việc phân tích các thành phần chức năng và cách chúng tương tác với nhau trong kiến trúc microservice. Dưới đây là một phân rã dựa trên Django của hệ thống e-commerce:

### 1. Dịch vụ Quản lý Sản phẩm (Product Management Service):

- **Mô tả:** Dịch vụ này quản lý thông tin về sản phẩm trên nền tảng e-commerce.
- **Chức năng:**
  - Quản lý danh sách sản phẩm, bao gồm thêm, sửa đổi, xóa sản phẩm.
  - Quản lý danh mục sản phẩm và các thuộc tính của chúng.
- **Công nghệ sử dụng:** Django ORM, Django Admin Interface.

### 2. Dịch vụ Quản lý Đơn hàng (Order Management Service):

- **Mô tả:** Dịch vụ này điều phối các hoạt động liên quan đến quản lý đơn hàng của khách hàng.
- **Chức năng:**
  - Xử lý đơn hàng: xác nhận, hủy bỏ, cập nhật trạng thái.
  - Quản lý thông tin vận chuyển và thanh toán.
- **Công nghệ sử dụng:** Django ORM, Django Admin Interface.

### 3. Dịch vụ Thanh toán (Payment Service):

- **Mô tả:** Dịch vụ này xử lý các giao dịch thanh toán an toàn và đáng tin cậy.
- **Chức năng:**
  - Xác thực và xử lý thanh toán từ các phương thức thanh toán khác nhau.
  - Gửi thông báo thanh toán và xử lý kết quả giao dịch.
- **Công nghệ sử dụng:** Django REST Framework, các cổng thanh toán bên thứ ba.

### 4. Dịch vụ Quản lý Người dùng (User Management Service):

- **Mô tả:** Dịch vụ này quản lý thông tin về người dùng và tài khoản trên nền tảng e-commerce.
- **Chức năng:**
  - Đăng ký, đăng nhập, quản lý thông tin cá nhân của người dùng.
  - Xác thực và quản lý vai trò người dùng.
- **Công nghệ sử dụng:** Django REST Framework, Django User Authentication.

### 5. Dịch vụ Quản lý Giỏ hàng (Cart Management Service):

- **Mô tả:** Dịch vụ này quản lý các mục hàng được thêm vào giỏ hàng của người dùng.
- **Chức năng:**

- Thêm, cập nhật, xóa mục hàng trong giỏ hàng của người dùng.
- Tính toán tổng giá trị của giỏ hàng.
- **Công nghệ sử dụng:** Django ORM, Django REST Framework.

#### 6. Dịch vụ Vận Chuyển (Shipping Service):

- **Mô tả:** Dịch vụ này quản lý các thông tin vận chuyển và giao hàng cho đơn hàng của khách hàng.
- **Chức năng:**
  - Xác định các phương thức vận chuyển khả dụng và tính phí vận chuyển dựa trên địa chỉ giao hàng.
  - Tích hợp với các nhà vận chuyển bên thứ ba để lấy thông tin vận chuyển và tạo mã vận đơn.
  - Cập nhật trạng thái vận chuyển và thông tin vận đơn cho khách hàng.
- **Công nghệ sử dụng:** Django ORM, Django REST Framework, API của các nhà vận chuyển.

Hệ thống được phân rã ra thành các service và được triển khai như sau:

Service	CSDL sử dụng	Tên DATABASE	Port
App	---	---	8000
user service	MySql	user	8001
cart service	MySql	cart	8002
order service	MongoDB	order	8003
payment service	PostgreSQL	payment	8005
shipment service	PostgreSQL	shipment	8006
product service	MongoDB	product	8008
search service	---	---	8009

## 2.4. Kết luận

Hệ thống ecomSys dựa trên Django được phân rã thành các dịch vụ độc lập, mỗi dịch vụ thực hiện một phần công việc cụ thể trong quá trình hoạt động của hệ thống. Sự sắp xếp này giúp tăng tính linh hoạt, hiệu suất và quản lý của hệ thống. Dưới đây là những điểm chính trong kết luận:

1. **Kiến trúc Microservice:** ecomSys sử dụng mô hình kiến trúc microservice, cho phép phân chia hệ thống thành các dịch vụ độc lập, có khả năng phát triển, triển khai và quản lý riêng biệt.
2. **Django Framework:** Việc sử dụng Django làm nền tảng cho hệ thống mang lại sự linh hoạt và hiệu quả trong việc phát triển ứng dụng web. Django cung cấp các công cụ mạnh mẽ như ORM, Admin Interface và REST Framework.
3. **Tích hợp dịch vụ:** Các dịch vụ trong hệ thống, bao gồm quản lý sản phẩm, đơn hàng, thanh toán, người dùng, giỏ hàng, thống kê và đánh giá, được tích hợp với nhau thông qua giao tiếp API và cơ sở dữ liệu chung.
4. **Dịch vụ mới:** Bằng cách thêm dịch vụ vận chuyển, hệ thống ecomSys trở nên hoàn chỉnh hơn, cung cấp cho người dùng trải nghiệm mua sắm toàn diện từ việc chọn sản phẩm cho đến nhận hàng.
5. **Mở rộng và Tùy chỉnh:** Kiến trúc microservice cho phép hệ thống dễ dàng mở rộng bằng cách thêm hoặc loại bỏ các dịch vụ một cách độc lập. Đồng thời, Django cung cấp khả năng tùy chỉnh linh hoạt cho từng dịch vụ.

Với kiến trúc này, ecomSys không chỉ mang lại trải nghiệm mua sắm trực tuyến tốt cho người dùng mà còn giúp doanh nghiệp dễ dàng mở rộng và quản lý hệ thống một cách hiệu quả.

## **Chương 3: Thiết kế và xây dựng hệ ecomSys**

### **3.1. Giới thiệu**

Thiết kế và xây dựng hệ thống ecomSys là quá trình phức tạp và đòi hỏi sự kỹ lưỡng từ việc lập kế hoạch ban đầu đến triển khai và duy trì sau này. Dưới đây là một sơ lược về quá trình thiết kế và xây dựng hệ thống ecomSys:

- Phân tích và Thu thập Yêu cầu:
  - Xác định mục tiêu kinh doanh và yêu cầu của khách hàng.
  - Thu thập thông tin về sản phẩm, người dùng, quản lý đơn hàng, vận chuyển, thanh toán và các yêu cầu khác.
  - Phân tích cạnh tranh và nghiên cứu thị trường để hiểu rõ về yêu cầu và xu hướng.
- Thiết kế Hệ thống:
  - Thiết kế Kiến trúc:
    - Xác định kiến trúc hệ thống, bao gồm các dịch vụ chính và cách chúng tương tác với nhau.
    - Lựa chọn mô hình kiến trúc phù hợp, chẳng hạn như kiến trúc microservice.
  - Thiết kế Cơ sở Dữ liệu:



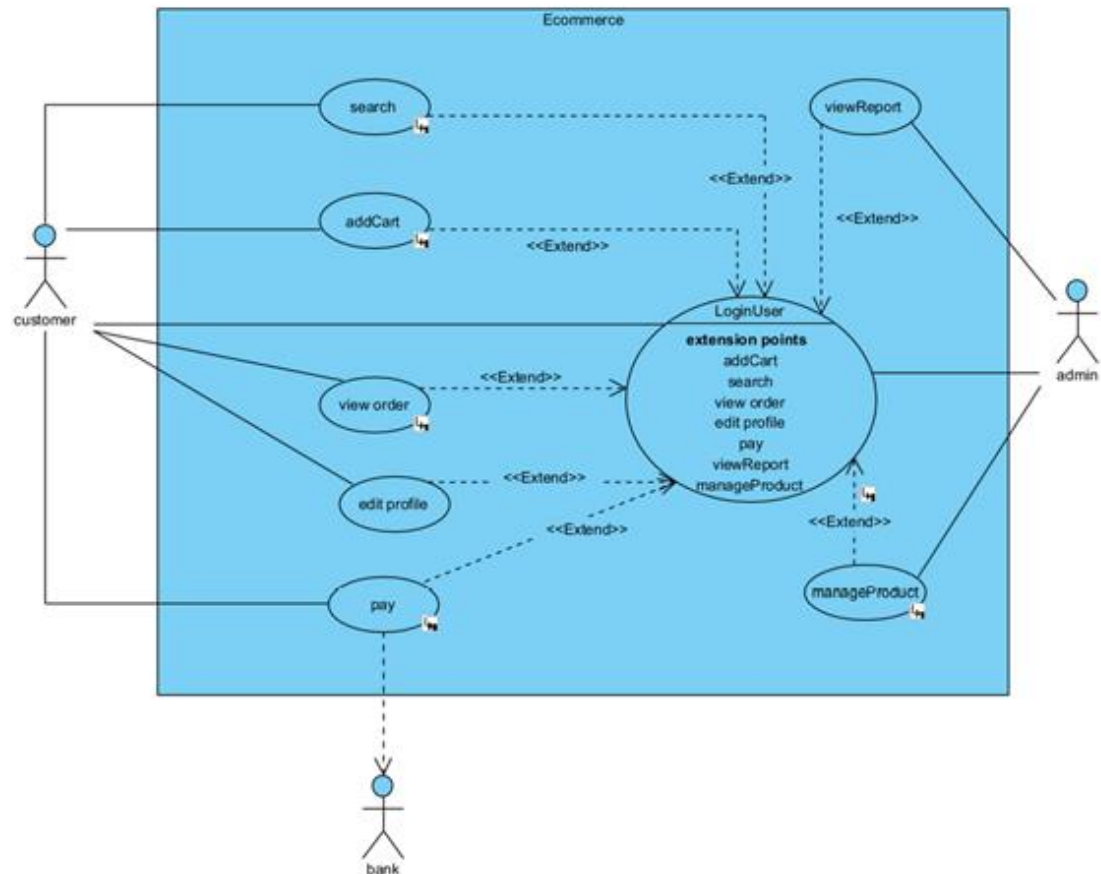
- Thiết kế cấu trúc cơ sở dữ liệu dựa trên yêu cầu của hệ thống và mối quan hệ giữa các thực thể.
- Lựa chọn loại cơ sở dữ liệu phù hợp, như PostgreSQL, MySQL hoặc MongoDB.
- Thiết kế Giao diện Người dùng:
  - Thiết kế giao diện người dùng dựa trên trải nghiệm người dùng dự kiến và các yêu cầu về thiết kế.
  - Tạo wireframes và mockups để minh họa giao diện trước khi triển khai.
- Phát triển và Triển khai:
  - Phát triển Ứng dụng:
    - Phát triển các dịch vụ và chức năng của hệ thống bằng cách sử dụng Django Framework và các công nghệ liên quan
    - Thực hiện kiểm thử đơn vị và kiểm thử tích hợp để đảm bảo tính ổn định và hiệu suất của hệ thống.
  - Triển khai và Kiểm tra:
    - Triển khai hệ thống trên môi trường sản xuất hoặc môi trường thử nghiệm, tùy thuộc vào yêu cầu của dự án.
    - Thực hiện kiểm tra chức năng và hiệu suất trước khi đưa hệ thống vào sử dụng thực tế.
- Duy trì và Nâng cấp:
  - Cung cấp hỗ trợ và bảo trì hệ thống sau khi triển khai để đảm bảo tính liên tục và ổn định.
  - Thực hiện các bản vá lỗi, cập nhật bảo mật và nâng cấp hệ thống theo yêu cầu của khách hàng và thị trường

Quá trình thiết kế và xây dựng hệ thống ecomSys đòi hỏi sự chuyên sâu và sự hợp tác chặt chẽ giữa các bộ phận khác nhau, bao gồm các nhà phát triển, nhà thiết kế, nhà quản lý dự án và nhà quản trị hệ thống. Đồng thời, việc áp dụng các phương pháp Agile và DevOps cũng giúp tăng cường quá trình phát triển và triển khai của dự án.

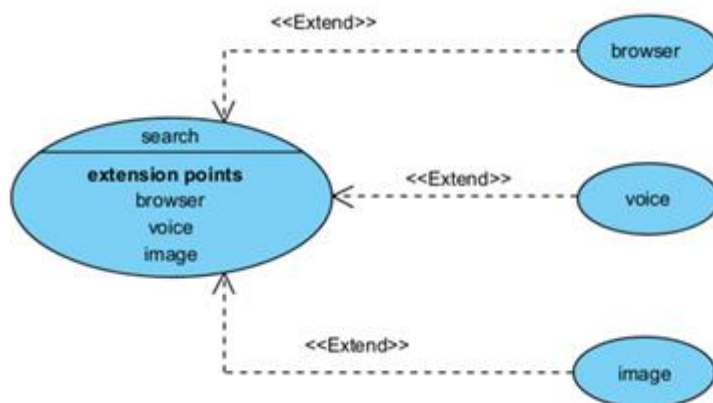
## 3.2. Các biểu đồ đặc tả hệ ecomSys

### 3.2.1. Biểu đồ use case

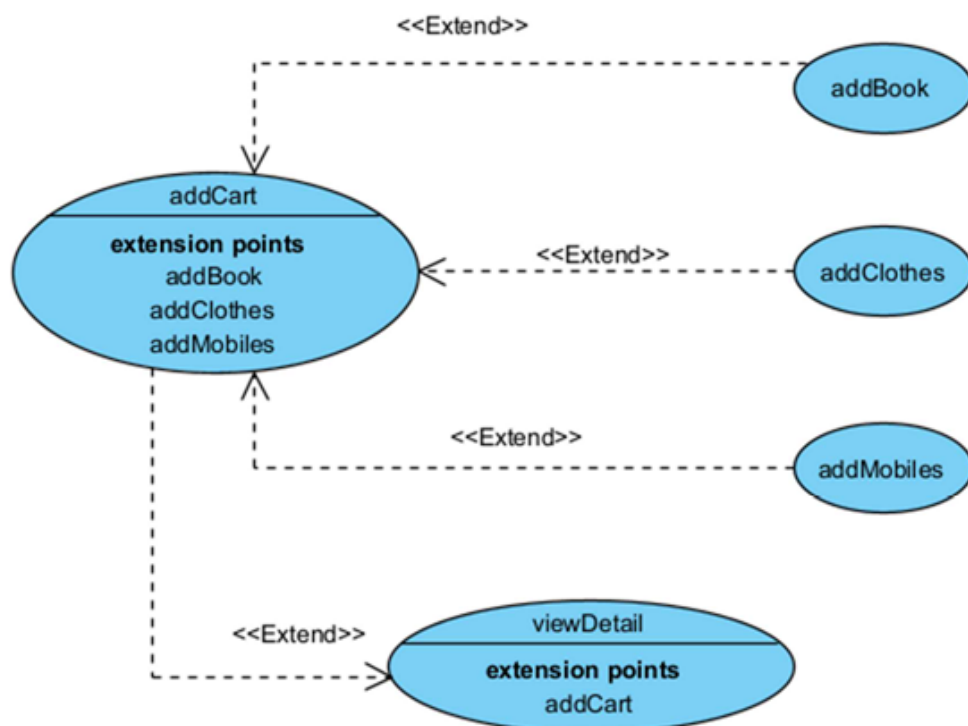
- *UseCase tổng quan*



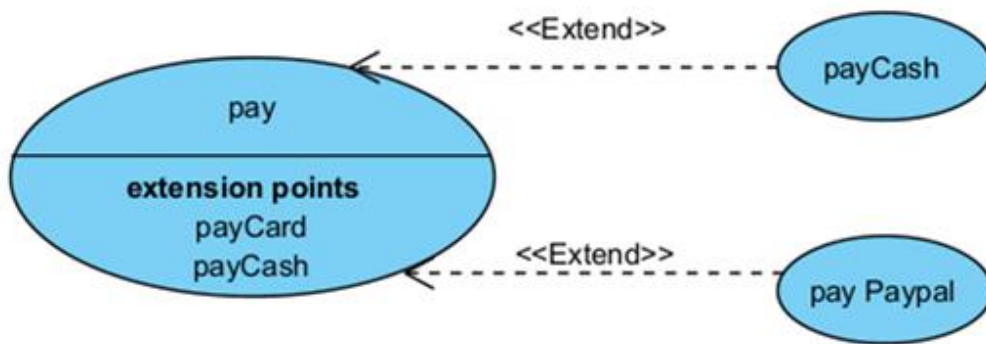
- *UseCase chi tiết – Search*



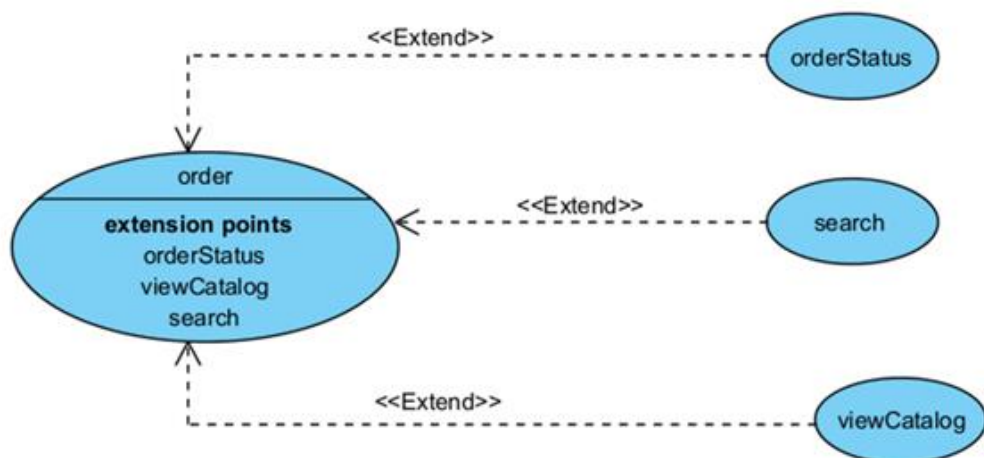
- *UseCase chi tiết – AddCart*



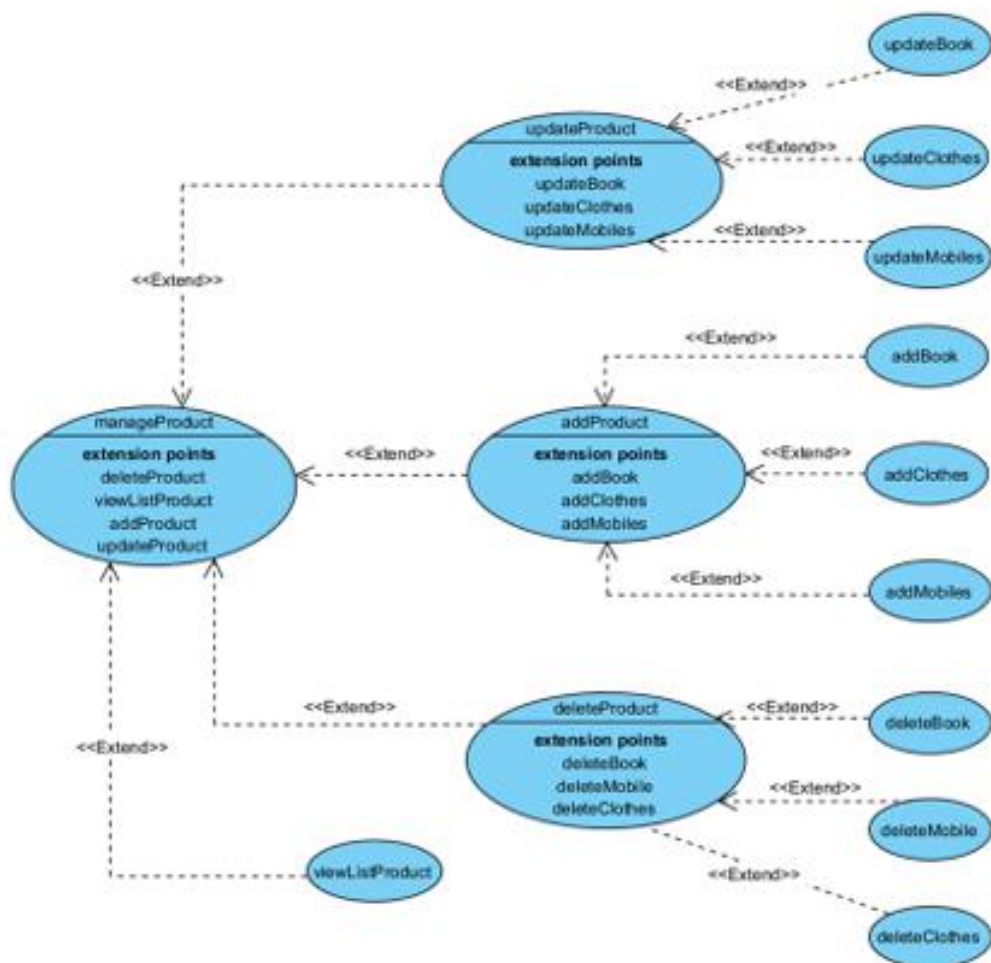
- *UseCase chi tiết – Pay*



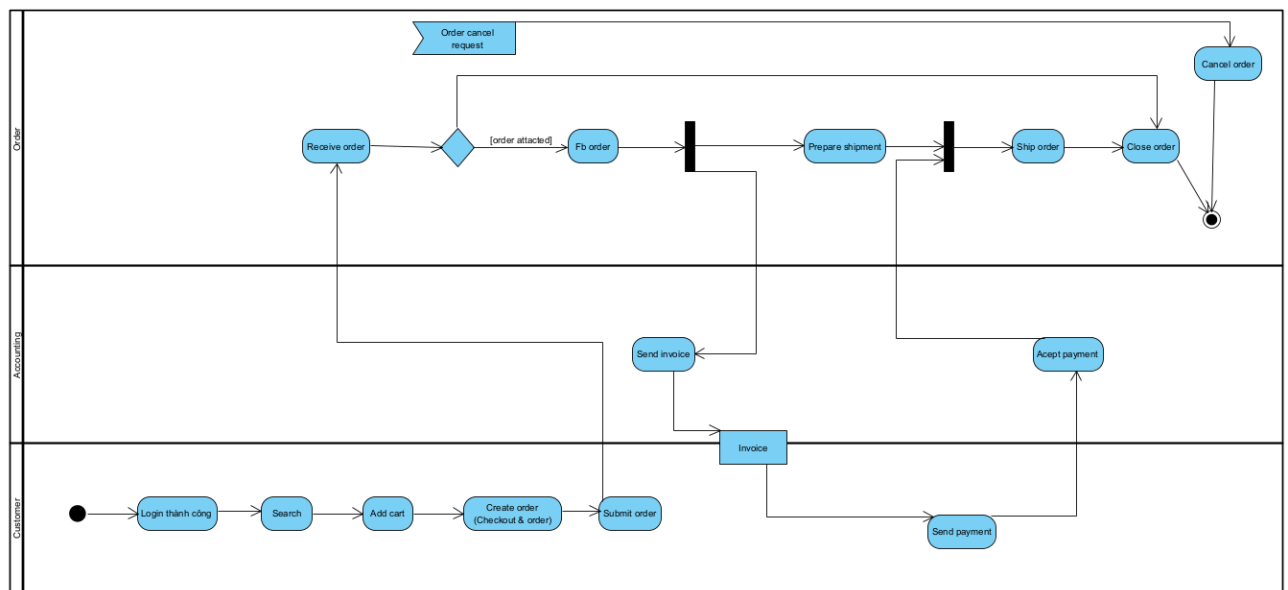
- *UseCase chi tiết – Order*



- *UseCase chi tiết – Manage Product*

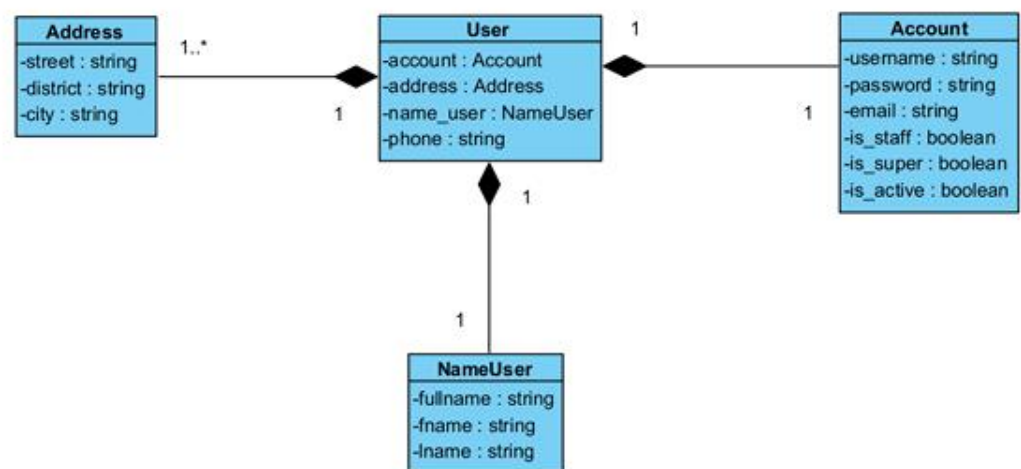


### 3.2.2. Biểu đồ hoạt động/swimlane

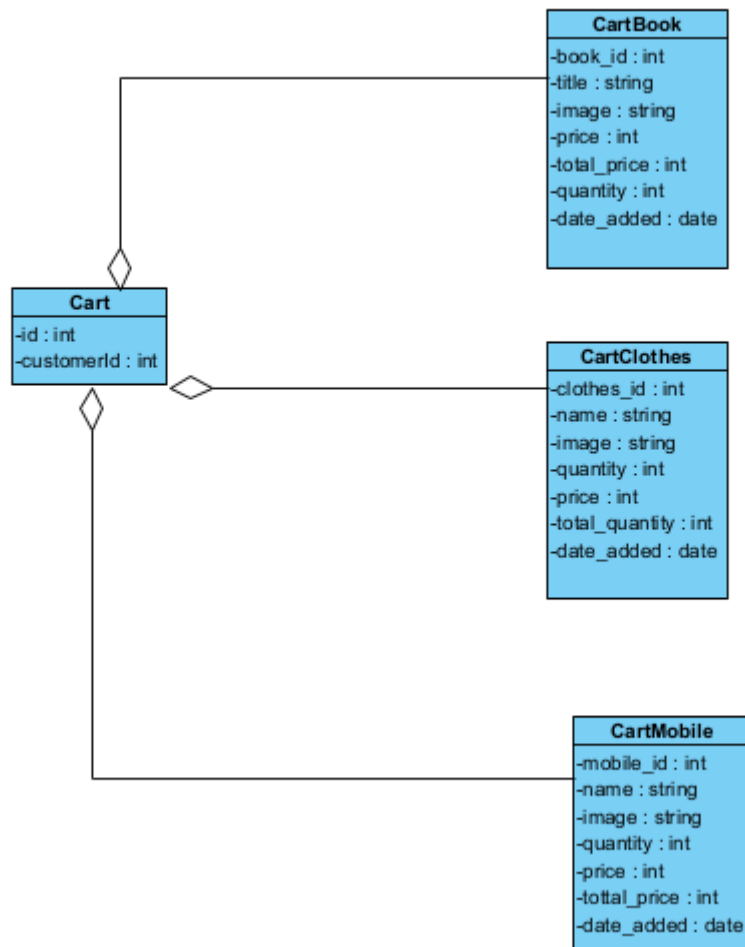


### 3.2.3. Biểu đồ lớp phân tích cho từng service

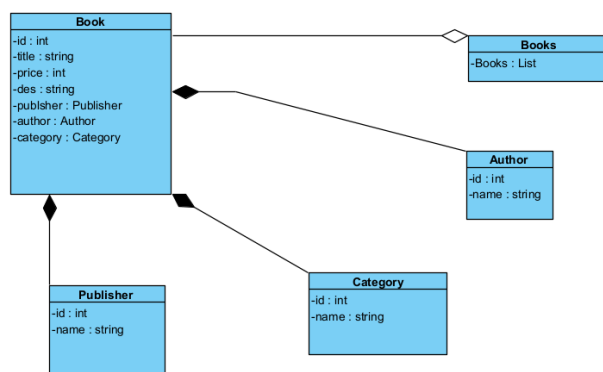
#### 3.2.3.1. User



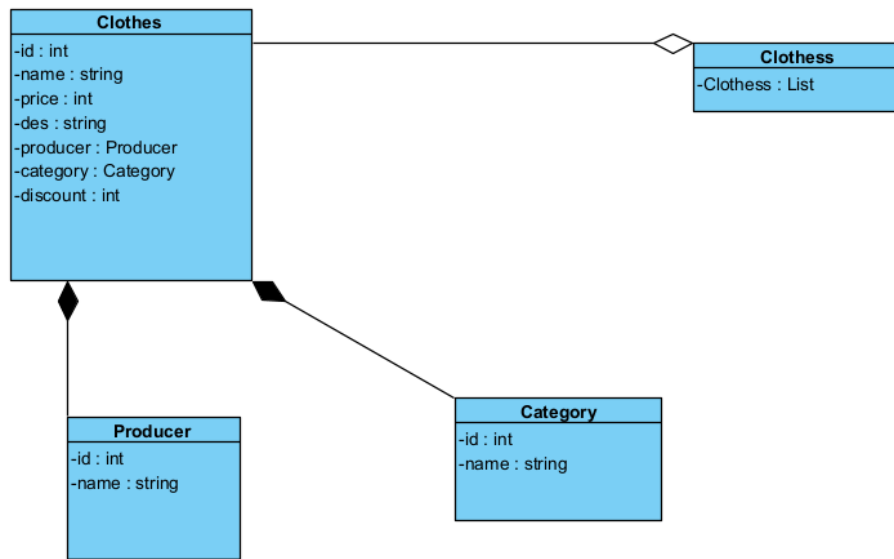
#### 3.2.3.2. Cart



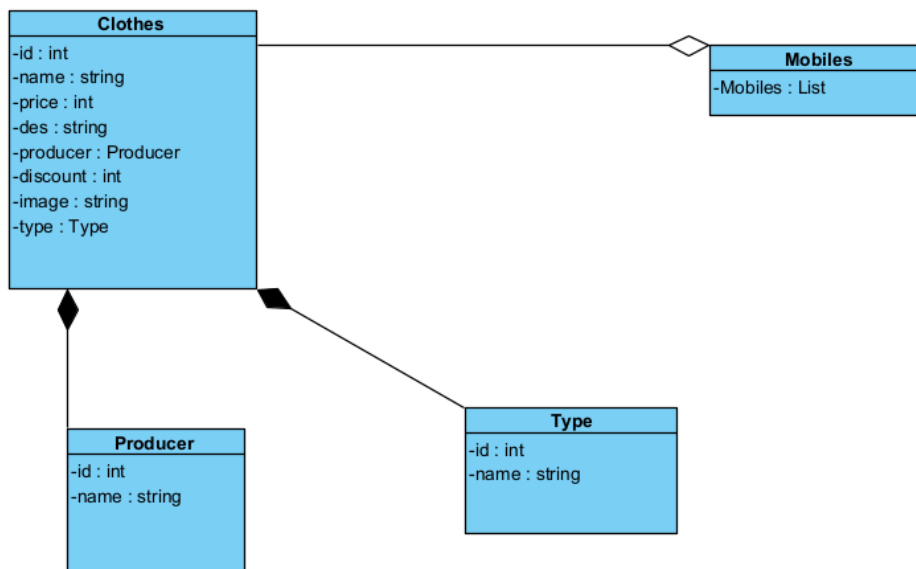
### 3.2.3.3. Book



### 3.2.3.4. Clothes

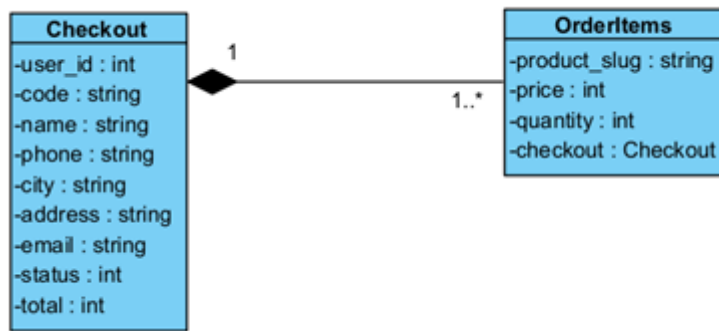


### 3.2.3.5. Mobile

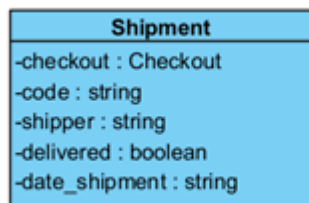


### 3.2.3.6. Order

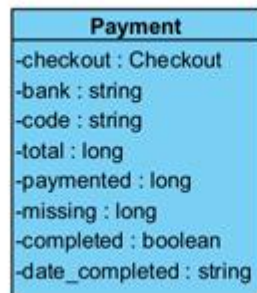




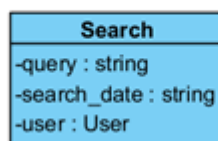
#### 3.2.3.7. Shipment



#### 3.2.3.8. Payment



#### 3.2.3.9. Search



### 3.2.4. Mô hình Data – kiểu/công nghệ dữ liệu

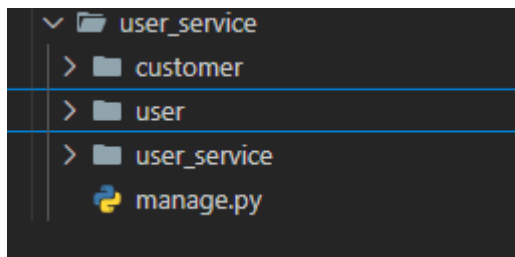
### 3.2.5. Biểu đồ lớp thiết kế cho từng service

## 3.3. Xây dựng hệ ecomSys

Quy trình tạo một service:

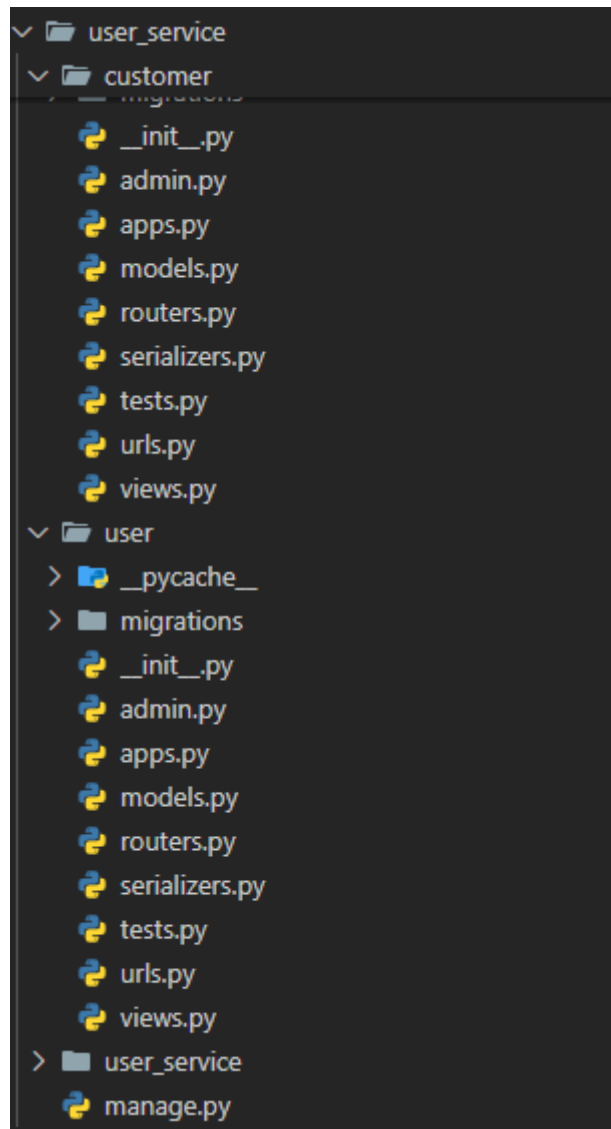
1. Tạo app
2. Cấu hình file setting tới cơ sở dữ liệu tương ứng
3. Tạo model
4. Makemigrations
5. Migrate
6. Tạo serializer
7. Tạo viewsets
8. Tạo REST API
9. Cấu hình URL cho rest api

### 3.3.1. Module (service/project) user



```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'user',  
    'customer',  
]
```

```
DATABASES = {  
    "default": {  
        "ENGINE": "django.db.backends.mysql",  
        "NAME": "user",  
        "USER": "root",  
        "PASSWORD": "12345",  
        "HOST": "localhost",  
        "PORT": "3306",  
    },  
    'mongodb': {  
        'ENGINE': 'djongo',  
        'NAME': 'test1',  
        'HOST': 'localhost',  
        'PORT': 27017,  
    },  
}
```

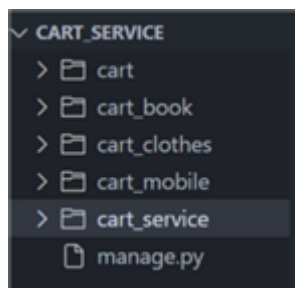


```

user > models.py > ...
1  from django.db import models
2
3  class NguoiDung(models.Model):
4      name = models.CharField(max_length=255)
5      address = models.CharField(max_length=255)
6      mobile = models.CharField(max_length=12)
7      role = models.CharField(max_length=20)
8      username = models.CharField(max_length=255)
9      password = models.CharField(max_length=255)
10
11     class Meta:
12         db_table = 'nguoidung'
13         app_label = 'user'
14
customer > models.py > ...
1  from django.db import models
2  from user.models import NguoiDung
3
4  class Customer(NguoiDung):
5      delivery_address = models.CharField(max_length=255)
6
7      class Meta:
8          db_table = 'customer'
9          app_label = 'customer'
10

```

### 3.3.2. Module Cart



```

from django.db import models

class Cart(models.Model):
    customer_id = models.IntegerField()

    class Meta:
        db_table = 'cart'

```

```

from django.db import models
from cart.models import Cart

class CartBook(models.Model):
    book_id = models.IntegerField()
    title = models.CharField(max_length=100)
    image = models.CharField(max_length=255)
    price = models.IntegerField()
    total_price = models.IntegerField()
    quantity = models.IntegerField()
    date_added = models.DateField(auto_now_add=True)
    cart = models.ForeignKey(Cart, related_name='cart_books', on_delete=models.CASCADE)

    class Meta:
        db_table = 'cart_book'

```

```

cart_mobile > models.py > CartMobile > Meta > db_table
1 from django.db import models
2 from cart.models import Cart
3
4 class CartMobile(models.Model):
5     mobile_id = models.IntegerField()
6     name = models.CharField(max_length=100)
7     image = models.CharField(max_length=255)
8     quantity = models.IntegerField()
9     price = models.IntegerField()
10    total_price = models.IntegerField()
11    date_added = models.DateField(auto_now_add=True)
12    cart = models.ForeignKey(Cart, related_name='cart_mobilis', on_delete=models.CASCADE)
13
14    class Meta:
15        db_table = 'cart_mobilis'

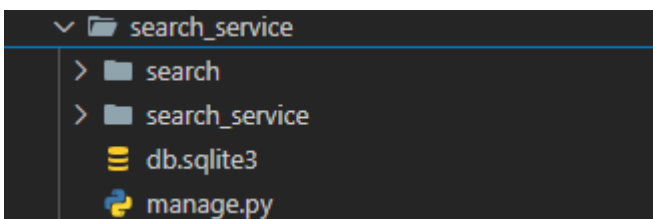
```

```

cart_clothes > models.py > CartClothes > quantity
1 from django.db import models
2 from cart.models import Cart
3
4 class CartClothes(models.Model):
5     clothes_id = models.IntegerField()
6     name = models.CharField(max_length=100)
7     image = models.CharField(max_length=255)
8     quantity = models.IntegerField()
9     price = models.IntegerField()
10    total_price = models.IntegerField()
11    date_added = models.DateField(auto_now_add=True)
12    cart = models.ForeignKey(Cart, related_name='cart_clothes', on_delete=models.CASCADE)
13
14    class Meta:
15        db_table = 'cart_clothes'

```

### 3.3.3. Module search



```

def search_keyword(request):
    keyword = request.GET['keyword']

    async def fetch_data(url):
        async with httpx.AsyncClient() as client:
            response = await client.get(url)
            return response.json()

    async def get_response():
        books_task = asyncio.create_task(fetch_data(f'http://localhost:8008/book/?search={keyword}'))
        clothes_task = asyncio.create_task(fetch_data(f'http://localhost:8008/clothes/?search={keyword}'))
        mobiles_task = asyncio.create_task(fetch_data(f'http://localhost:8008/mobile/?search={keyword}'))
        cart_book_task = asyncio.create_task(fetch_data('http://localhost:8008/category-book/'))
        cart_clothes_task = asyncio.create_task(fetch_data('http://localhost:8008/category-clothes/'))
        cart_mobile_task = asyncio.create_task(fetch_data('http://localhost:8008/category-mobile/'))
        list_books = await books_task
        list_clothes = await clothes_task
        list_mobiles = await mobiles_task
        cart_book = await cart_book_task
        cart_clothes = await cart_clothes_task
        cart_mobile = await cart_mobile_task
        return {
            'list_books': list_books,
            'list_clothes': list_clothes,
            'list_mobiles': list_mobiles,
            'cart_book': cart_book,
            'cart_clothes': cart_clothes,
            'cart_mobile': cart_mobile,
        }

```

### 3.3.4. Module product

#### 3.3.4.1. App Book

```

from django.db import models

class CategoryBook(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        db_table = 'category'
        app_label = 'book'

```

```

    def __str__(self):
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=255)
    email = models.CharField(max_length=255)

    class Meta:
        db_table = 'author'
        app_label = 'book'

    def __str__(self):
        return self.name

class Publisher(models.Model):
    name = models.CharField(max_length=255)
    email = models.CharField(max_length=255)

    class Meta:
        db_table = 'publisher'
        app_label = 'book'

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=255)
    image = models.ImageField(upload_to='book', blank=True, null=True)
    brand = models.CharField(max_length=50)
    price = models.IntegerField()
    old_price = models.IntegerField()
    description = models.TextField(max_length=1000)
    quantity = models.IntegerField()
    is_active = models.BooleanField()
    is_bestseller = models.BooleanField()
    is_feature = models.BooleanField()
    create_at = models.DateField(auto_now_add=True)
    update_at = models.DateField(auto_now=True)
    category = models.ForeignKey(CategoryBook, related_name='books',
on_delete=models.CASCADE)
    author = models.ForeignKey(Author, related_name='books',
on_delete=models.CASCADE)
    publisher = models.ForeignKey(Publisher, related_name='books',
on_delete=models.CASCADE)

```

```

class Meta:
    db_table = 'book'
    app_label = 'book'

```

```

book > D routers.py > ts BookRouter > allow_relation
1 class BookRouter:
2     # truy vấn đọc (SELECT)
3     def db_for_read(self, model, **hints):
4         if model._meta.app_label == 'book':
5             return 'default'
6         return None
7     # truy vấn ghi (INSERT, UPDATE, DELETE)
8     def db_for_write(self, model, **hints):
9         if model._meta.app_label == 'book':
10            return 'default'
11        return None
12
13    # cho phép quan hệ (relationship) giữa hai đối tượng (obj1 và obj2)
14    def allow_relation(self, obj1, obj2, **hints):
15        if (
16            obj1._meta.app_label == 'book' and
17            obj2._meta.app_label == 'book'
18        ):
19            return True
20        return None
21
22    # thực hiện các migration trên database
23    def allow_migrate(self, db, app_label, model_name=None, **hints):
24        if app_label == 'book':
25            return db == 'default'
26        return None

```



### 3.3.4.2. App Clothes

```
from django.db import models

class CategoryClothes(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        db_table = 'category'
        app_label = 'clothes'
```

```
def __str__(self):
    return self.name

class Clothes(models.Model):
    name = models.CharField(max_length=255)
    image = models.ImageField(upload_to='book', blank=True, null=True)
    brand = models.CharField(max_length=50)
    price = models.IntegerField()
    old_price = models.IntegerField()
    description = models.TextField(max_length=1800)
    quantity = models.IntegerField()
    is_active = models.BooleanField()
    is_bestseller = models.BooleanField()
    is_feature = models.BooleanField()
    create_at = models.DateTimeField(auto_now_add=True)
    update_at = models.DateTimeField(auto_now=True)
    category = models.ForeignKey(CategoryClothes, related_name='clothes',
on_delete=models.CASCADE)

    class Meta:
        db_table = 'clothes'
        app_label = 'clothes'
```

```
clothes > routers.py > ClothesRouter > db_for_write
1 ~ class ClothesRouter:
2     # truy vấn đọc (SELECT)
3     def db_for_read(self, model, **hints):
4         if model._meta.app_label == 'clothes':
5             return 'mongodb'
6         return None
7     # truy vấn ghi (INSERT, UPDATE, DELETE)
8     def db_for_write(self, model, **hints):
9         if model._meta.app_label == 'clothes':
10            return 'mongodb'
11        return None
12
13 ~ # cho phép quan hệ (relationship) giữa hai đối tượng (obj1 và obj2)
14 ~ def allow_relation(self, obj1, obj2, **hints):
15 ~     if (
16         obj1._meta.app_label == 'clothes' and
17         obj2._meta.app_label == 'clothes'
18     ):
19         return True
20     return None
21
22 ~ # thực hiện các migration trên database
23 ~ def allow_migrate(self, db, app_label, model_name=None, **hints):
24 ~     if app_label == 'clothes':
25         return db == 'mongodb'
26     return None
```

### 3.3.4.3. App Mobile

```

from django.db import models

class CategoryMobile(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        db_table = 'category'
        app_label = 'mobile'

    def __str__(self):
        return self.name

class Mobile(models.Model):
    name = models.CharField(max_length=255)

```

```

image = models.ImageField(upload_to='book', blank=True, null=True)
brand = models.CharField(max_length=50)
price = models.IntegerField()
old_price = models.IntegerField()
description = models.TextField(max_length=1000)
quantity = models.IntegerField()
is_active = models.BooleanField()
is_bestseller = models.BooleanField()
is_feature = models.BooleanField()
create_at = models.DateTimeField(auto_now_add=True)
update_at = models.DateTimeField(auto_now=True)
category = models.ForeignKey(CategoryMobile, related_name='mobies',
on_delete=models.CASCADE)

    class Meta:
        db_table = 'mobile'
        app_label = 'mobile'

```

```

mobile > routers.py > MobileRouter
1 class MobileRouter:
2     # truy vấn đọc (SELECT)
3     def db_for_read(self, model, **hints):
4         if model._meta.app_label == 'mobile':
5             return 'postgres'
6         return None
7     # truy vấn ghi (INSERT, UPDATE, DELETE)
8     def db_for_write(self, model, **hints):
9         if model._meta.app_label == 'mobile':
10            return 'postgres'
11        return None
12
13    # cho phép quan hệ (relationship) giữa hai đối tượng (obj1 và obj2)
14    def allow_relation(self, obj1, obj2, **hints):
15        if (
16            obj1._meta.app_label == 'mobile' and
17            obj2._meta.app_label == 'mobile'
18        ):
19            return True
20        return None
21
22    # thực hiện các migration trên database
23    def allow_migrate(self, db, app_label, model_name=None, **hints):
24        if app_label == 'mobile':
25            return db == 'postgres'
26        return None

```

### 3.3.5. Module payment

### 3.3.6. Module shipment

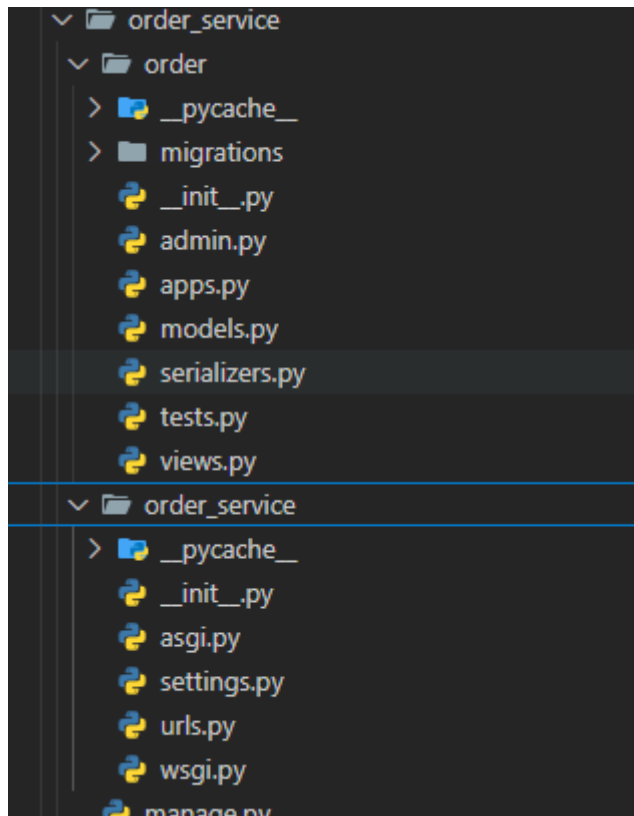
```
shipment > models.py > Shipment > Meta > db_table
1  from django.db import models
2
3  class Shipment(models.Model):
4      name = models.CharField(max_length=100)
5      mobile = models.CharField(max_length=12)
6      delivery_address = models.CharField(max_length=255)
7      order_id = models.IntegerField()
8      shipment_status = models.CharField(max_length=255)
9      payment_status = models.CharField(max_length=255)
10     time = models.DateTimeField(auto_now_add=True)
11
12     class Meta:
13         db_table = 'shipment'
```

### 3.3.7. Module order

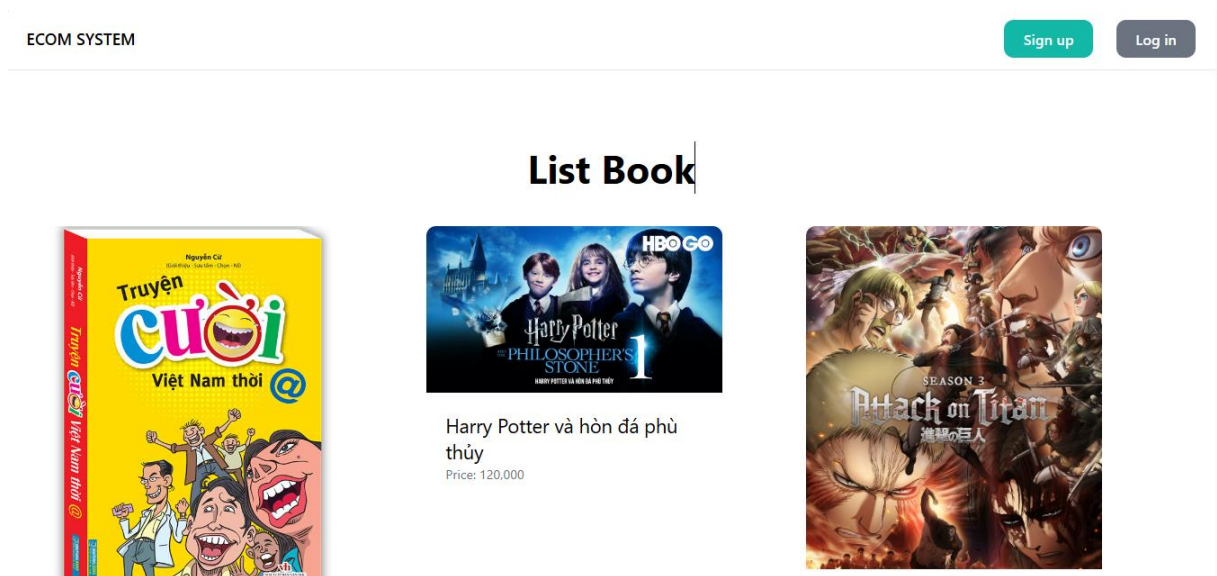
```
from django.db import models

class Order(models.Model):
    username = models.CharField(max_length=255)
    mobile = models.CharField(max_length=12)
    delivery_address = models.CharField(max_length=255)
    order_product = models.JSONField()
    total_price = models.IntegerField()
    date = models.DateTimeField(auto_now_add=True)
    isDone = models.BooleanField(default=False)
    approved = models.BooleanField(default=False)
    customer_id = models.IntegerField()

    class Meta:
        db_table = 'order'
```



### 3.4. Thiết kế giao diện



## Đăng nhập hệ thống

Số điện thoại (\*)

duymanh

Mật khẩu (\*)

.....



Đăng nhập

Về trang chủ

### Giao diện login

ECOM SYSTEM

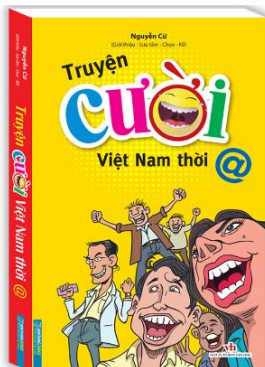
Search

Cart

Your Order

Logout

## List Book

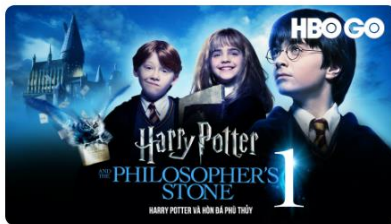


Harry Potter và hòn đá phù thủy

Price: 120,000



### Giao diện trang chủ đã login



## Harry Potter và hòn đá phù thủy

Price: 120,000

Brand: Wanner Bross

Author: Nguyen Van A

Category: Trinh Tham

Publisher: Kim Dong

Amount: 100

Description:

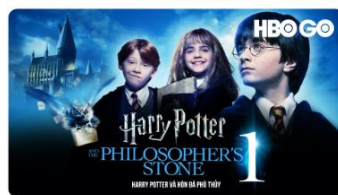
No description

Add to cart

*Giao diện hiển thị 1 product*



## Harry Potter và hòn đá phù thủy



Harry Potter và hòn đá phù thủy

Total Amount: 100

Chọn số lượng:


Add to cart

*Giao diện add to cart*

ECOM SYSTEM

SearchCartYour OrderLogout

Your cart



Harry Potter và hòn đá phù thủy  
Đơn giá 120,000 đ  
Số lượng: 1  
Thành tiền: 120,000 đ

Tổng tiền: 120,000 đ


Mua ngay

Giao diện cart

ECOM SYSTEM

SearchCartYour OrderLogout

Complete your order



Harry Potter và hòn đá phù thủy  
Đơn giá 120,000 đ  
Số lượng: 1  
Thành tiền: 120,000 đ

Tổng tiền: 120,000 đ

Type of purchase

Cash

Your address

default 0944516726

Your delivery address

default thôn Thọ Sơn 2

Confirm order

Giao diện order

You've completed your order

[back to home](#)

Quay lại

20/05/2024

Total price: 120,000 đ Unapproved

### 3.5. Thiết kế database

Sử dụng Mysql, MongoDB, PostgreSQL

Router đối với những service dùng nhiều Database

### 3.6. Kết luận

Trong quá trình thực hiện dự án eCommerce, em đã phân tích, thiết kế và phát triển một hệ thống mua sắm trực tuyến toàn diện, sử dụng Django làm framework chính.. Dưới đây là những điểm chính mà chúng ta đã đạt được trong quá trình này:

#### 1. Phân rã và Kiến trúc Microservice:

- Hệ thống eCommerce đã được phân rã thành các dịch vụ độc lập như quản lý sản phẩm, đơn hàng, thanh toán, người dùng, giỏ hàng, cùng với dịch vụ vận chuyển.
- Việc sử dụng kiến trúc microservice giúp tăng tính linh hoạt, khả năng mở rộng và khả năng bảo trì của hệ thống.

#### 2. Sử dụng Django:

- Django framework cung cấp một nền tảng mạnh mẽ cho phát triển ứng dụng web, với các tính năng như ORM, Admin Interface và Django REST Framework.
- Việc tận dụng các công cụ và thư viện của Django giúp tăng tốc quá trình phát triển và giảm thiểu lỗi.

#### 3. Tích hợp và Quản lý Dữ liệu:

- Hệ thống được thiết kế để dễ dàng tích hợp với các dịch vụ bên ngoài, như cổng thanh toán và dịch vụ vận chuyển.
- Cơ sở dữ liệu được thiết kế và quản lý một cách hiệu quả để đảm bảo tính nhất quán và an toàn của dữ liệu.

#### 4. Trải nghiệm Người dùng:

- Giao diện người dùng được thiết kế thân thiện và dễ sử dụng, đảm bảo trải nghiệm mua sắm trực tuyến mượt mà và hiệu quả.
- Các chức năng như giỏ hàng, quản lý tài khoản và theo dõi đơn hàng giúp cải thiện sự hài lòng của khách hàng.

## Hướng Phát triển Tương Lai

- **Mở rộng Tính Năng:** Thêm các tính năng mới như chương trình khách hàng thân thiết, gợi ý sản phẩm dựa trên AI và phân tích dữ liệu người dùng để cá nhân hóa trải nghiệm mua sắm.
- **Tối ưu hóa Hiệu Suất:** Liên tục tối ưu hóa hệ thống để đảm bảo khả năng mở rộng và đáp ứng nhanh chóng trong mọi tình huống.
- **Bảo mật Nâng Cao:** Triển khai thêm các biện pháp bảo mật, như xác thực hai yếu tố và mã hóa dữ liệu nâng cao.